# go语言设计与实现-gc荐

## 编译原理

golang的源码是可以自己修改重新编译的。

抽象语法树 静态单赋值(SSA)

X86指令集 ARM指令集

编译器前端:承担着词法分析语法分析(生成抽象语法树),类型检查,中间码生成.

编译器后端:目标代码的生成和优化,翻译成中间码到机器码.

go源码包里面包含了不同指令集机器码生成所需的包,比如生成x86机器码的包, ARM机器码的包.

有了抽象语法树,那么类型检查的时候就会类似操作树一样什么left right node之类的了.

中间代码的存在是必要的,不能直接源码到机器码的.

中间代码的生成过程就是抽象语法树到SSA中间代码的转换过程.

指令集是软硬件之间的桥梁,所以指令集架构是计算机的抽象模型,所以指令集架构也是计算机架构,包括x86和ARM.

X86就是复杂指令集,ARM是精简指令集.由于复杂指令集指令很多,指令长度也就不同,需要消耗额外的性能去判断指令长度,而简单指令集是用固定长度的基础指令组合使用,所以更好,性能更好.

中间代码会转成汇编代码, 汇编代码经过汇编器就会转成机器码

静态语法树 - SSA中间代码 - 汇编代码 - 机器码

## 数据结构

那种通过下标创造的切片是一个新的切片,他也有自己的len cap属性,但是data还是指向原来的切片,所以原来的改了他也会改.

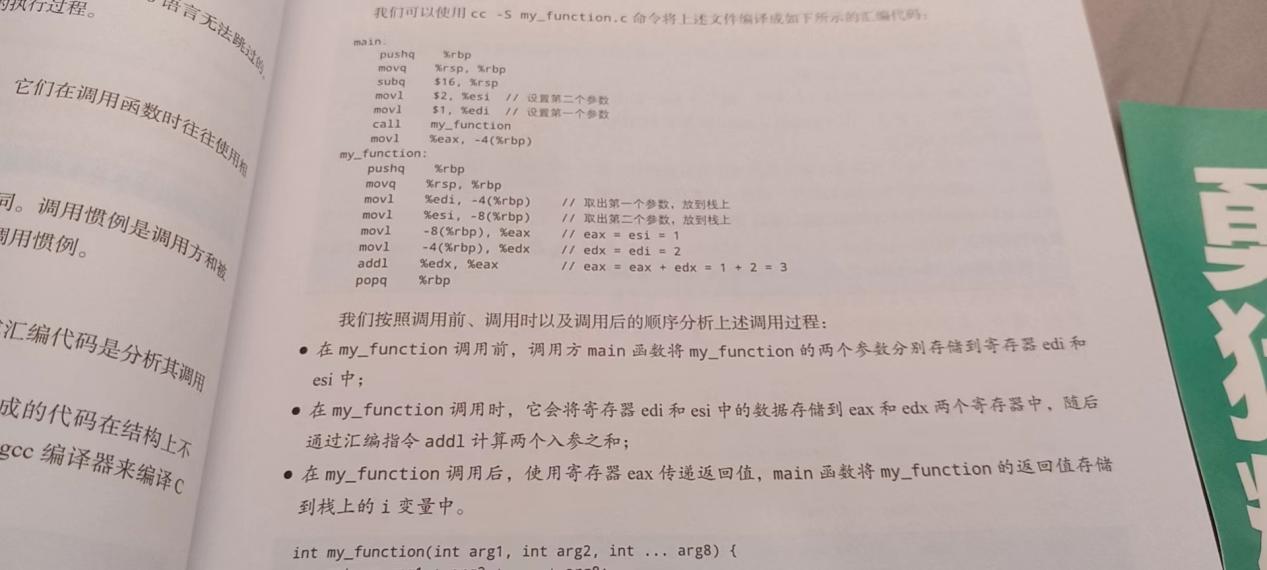
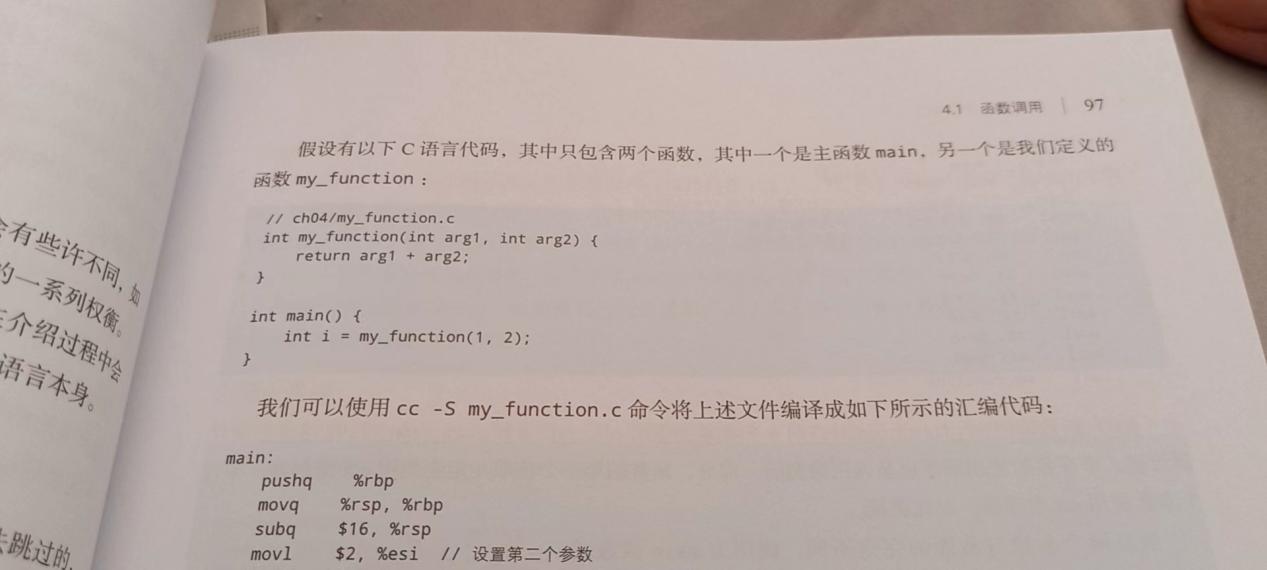
切片的扩容也会用到memove

Map todo

字符串的结构和切片的区别就是字符串没有cap字段.字符串的拼接也是需要用到copy函数的,而且新拼出来的字符串是个新的空间,不是追加, 所以不挺的拼接,那么就会不停copy越来越大的空间性能就会下降.

字符串是不可写的,转成[]byte是可写的,其实这个转换也是copy了一份,如果很大的话也是会损耗性能的.

## 函数调用



可以看到上面的汇编,确实是值传递,里面的函数会复制一份外面传进来的参数使用.

是这样的,外面的1和2的实参,是main函数放到寄存器中的,myfunc用的时候会从寄存器里面复制数据放到自己的栈里,所以是值传递.最后在用eax寄存器传递返回值,这个寄存器是专门用来传递返回值的,只有一个所以c语言只有一个返回值.

看来寄存器是有个数限制的,而且都有自己独特的名字.

c语言和go语言在设计函数调用选择不同的.

C语言使用寄存器和栈传递参数(如果参数数量小于6个全用的寄存器)

go使用栈传递参数和返回值.

c语言这样更快,cpu访问寄存器的速度比访问栈快几十倍.

go这样比较简单,不需要适配不同的寄存器,可以返回多个值,因为返回值也是在栈上,栈上多开空间不就可以返回多个了,但是寄存器却只有几个,就不能返回多个参数了.

结构体和数组是值传递, 所以大结构体和数组传递尽量用指针避免复制

切片和指针也是值传递,不过传递的是指针就还好.

类型这个东西在runtime包里定义成\_type.

里面有很多字段,有一个叫hash的字段,用来断言的时候判断是不是同一个类型.

直接通过反射修改一个整数的值是不行的.

同构反射获取Elm修改一个整数的指针修改整数是可以的.

## 语法

for循环的实现基本都是用到了汇编的

JUMP命令跳转 INCQ命令++ CMPQ命令比较.

defer就用func的方式就好,这样特性用的是函数的特性.

recover只能在defer中使用.

new的意思是 根据传递的类型申请一块空间,并返回该空间的指针.

为了能cancel父的时候把子也cancel了,所以cancelctx肯定有个地方存了自己的childmap.那么valuectx肯定也存了自己的parent.

## 并发

Mutx lock方法

先是cas操作看看能不能直接锁上,不能的话就自旋,再看能不能用cas锁上.如果还不能就通过信号量的方式租的获取锁了.

Unlock 先看能不能cas操作释放锁,不能的话就信号量.

所以golang也是有自旋锁的.

Waitgroup 的add和done就是去操作++ -- 一个数 wait就是不听的for循环读这个数是不是为0了,为0就特么跳出来.